Дәріс 10. Асинхронды программалау принциптері. C# тілінің асинхронды функциялары

Дәрістің мақсаты: Студенттерде асинхронды программалау қағидалары туралы және C# тілінің асинхронды функцияларын пайдалану туралы түсінік қалыптастыру.

Дәрісті меңгеру нәтижесінде студенттер келесі қабілеттерге ие болады:

- Асинхронды программалау қағидаларын түсіну;
- C# тілінің async, await кілттік сөздерін түсіну

Синхронды және асинхронды операциялар

Синхронды операция шақырушы абонентке қайтарар алдында өз жұмысын орындайды. Асинхрондық операция шақырушы абонентке қайтарылғаннан кейін өз жұмысын (көпшілігі немесе барлығы) орындай алады.

Жазу және шақыру әдістерінің көпшілігі синхронды болып табылады. Мысал: List<T>.Add, Console.WriteLine немесе Thread.Sleep.

Асинхронды әдістер аз таралған және параллелизмге бастама береді, себебі жұмыс шақырушы абонентке қатар жалғасады. Асинхронды әдістер әдетте тез (немесе дереу) шақырушы абонентке қайтарылады; осылайша, олар сондай-ақ тығыз емес тәсілдер деп аталады.

Асинхронды бағдарламалау қағидаты ұзын (немесе әлеуетті ұзақ) функцияларды асинхронды жазу болып табылады. Бұл ұзақ функцияларды синхронды жазудың әдеттегі тәсіліне қайшы келеді және осы функцияларды жаңа ағымнан немесе қажеттілігіне қарай параллелизмді енгізу үшін тапсырмадан кейіннен шақырады.

Асинхрондық тәсілдің айырмашылығы мынада: параллелизм функцияның сыртында емес, ұзақ уақыт орындалатын функцияның ішінде бастама алады. Бұл екі артықшылық береді:

- Енгізу-шығару параллелизмін ағындарды байланыстырмай іске асыруға болады ("TaskCompletionSource" көрсетілгендей), бұл ауқымдылық пен тиімділікті арттырады.
- Бай клиенттері бар қосымшалар ақырында жұмыс ағындарында кодтан аз болады, бұл ағындардың қауіпсіздігін жеңілдетеді.

Бұл өз кезегінде асинхронды бағдарламалауды пайдаланудың екі түрлі түріне әкеледі. Біріншісі - бұл қосымша жазбасы (әдетте сервер жағында), олар көп параллель енгізу-шығару операцияларымен тиімді жұмыс істейді. Бұл жерде мәселе ағындардың қауіпсіздігінде емес (әдетте жалпы ең аз жағдай бар), ағындардың тиімділігінде; атап айтқанда, желілік сұрау ағынын тұтынбайды. Осылайша, осы контексте тек енгізу-шығару операциялары ғана асинхронды үндестірудің артықшылықтарына ие болады. Асинхронды әдістер әдетте тез (немесе дереу) шақырушы абонентке қайтарылады; осылайша, олар сондай-ақ тығыз емес тәсілдер деп аталады.

C# тілінің асинхронды функциялары

Async және await түйінді сөздері асинхронды бағдарламалаудың "сантехникасын" қоспағанда, синхронды кодпен бірдей құрылымы мен қарапайымдылығы бар асинхронды кодты жазуға мүмкіндік береді.

Await кілт сөзі жалғастыруларды қосуды жеңілдетеді. Негізгі сценарийден бастап компилятор мыналарды кеңейтеді:

```
var result = await expression;
statement(s);
```

функционалдығы бойынша мынаған ұқсас:

```
var awaiter = expression.GetAwaiter();
awaiter.OnCompleted (() =>
{
var result = awaiter.GetResult();
statement(s);
});
```

Көрсету үшін қарапайым сандарды есептейтін және есептейтін бұрын жазылған асинхронды әдіске оралайық:

```
Task<int> GetPrimesCountAsync (int start, int count)
{
return Task.Run (() =>
ParallelEnumerable.Range (start, count).Count (n =>
Enumerable.Range (2, (int)Math.Sqrt(n)-1).All (i => n % i > 0)));
}
```

Әдісті шақыру:

```
int result = await GetPrimesCountAsync (2, 1000000);
Console.WriteLine (result);
```

Құрастыру үшін мыналарды қамтитын әдіске асинхронды модификаторды қосу қажет:

```
async void DisplayPrimesCount()
{
int result = await GetPrimesCountAsync (2, 1000000);
Console.WriteLine (result);
}
```

Async түрлендіргіш компиляторға осы әдісте біркелкі емес, await сөзі ретінде қарауды ұйғарады (бұл await идентификатор ретінде пайдалана алатын C# 5 дейін жазылған кодтың қатесіз жинақталуына кепілдік береді). Async түрлендіргіш void мәнін қайтаратын, Task немесе Task<TResult> мәнін қайтаратын әдістерге (және лямбда-өрнектерге) ғана қолданылуы мүмкін.

Асинхронды модификаторы бар әдістер асинхронды функциялар деп аталады, өйткені олардың өздері әдетте асинхронды болады. Неге екенін білу үшін асинхронды функцияның қалай орындалып жатқанын көрейік.

Await өрнегі анықталған кезде (әдетте) орындау шақырушы абонентке қайтарылады - итератордағы кірістілікті қайтару сияқты. Бірақ қайтар алдында орындау ортасы жалғасын

күтілетін тапсырмаға қосып, тапсырманы аяқтау кезінде орындау әдіске қайта ауысып, ол қалған жерде жалғасады. Тапсырманың сәтсіздігі кезінде оның ерекшелігі қайта құрылады, керісінше жағдайда оның қайтарылатын мәні күту өрнегіне беріледі. Біз жаңа зерттелген асинхронды әдістің қисынды кеңеюіне қарап, айтылғандардың бәрін жинақтай аламыз:

```
void DisplayPrimesCount()
{
var awaiter = GetPrimesCountAsync (2, 1000000).GetAwaiter();
awaiter.OnCompleted (() =>
{
int result = awaiter.GetResult();
Console.WriteLine (result);
});
}
```

Сіз күткен өрнек әдетте міндет болып табылады; алайда күтуші нысанды қайтаратын (IsCompleted іске асыратын және тиісті типтелген әдісі және bool сипаты бар) әдісі бар кез келген нысан компиляторға қанағаттандыратын болады. Біздің күту өрнегіміздің int түріне ие екенін ескеріңіз; бұл күтілетін өрнектің Task<int> болып табылатындығымен байланысты.

Генерикалық емес міндетті күту заңды болып табылады және:

```
await Task.Delay (5000);
Console.WriteLine ("Five seconds passed!");
```